A Parallel Approach to the AI of Connect-5

Wesley Chen ‘15 and Brandon Sim ‘15

Institute of Applied Computational Science, Harvard University

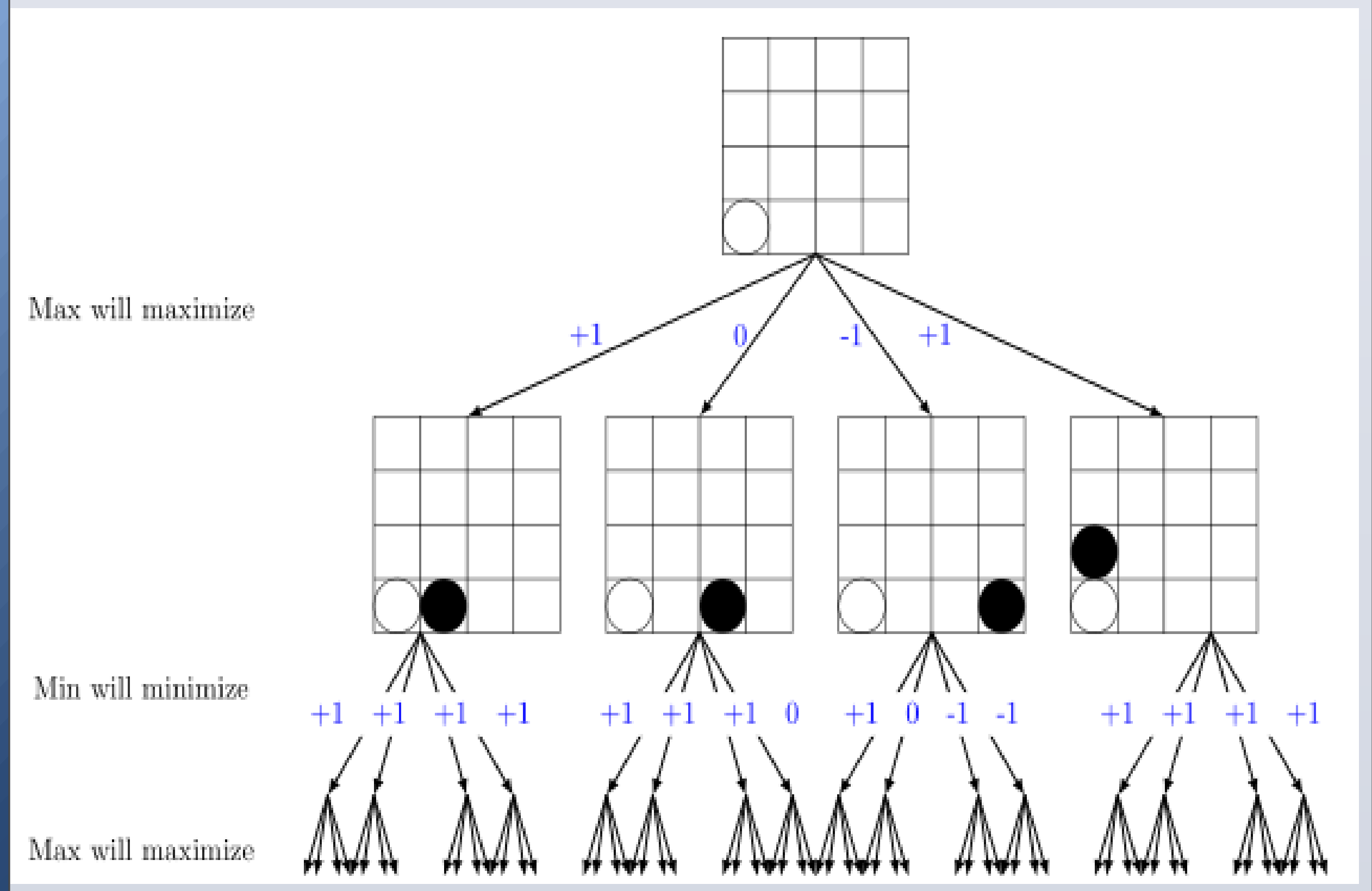
Introduction and Motivation

Connect-5 (also known as Gomoku) is a 2-player, zero-sum board game normally played on a 15x15 grid. The players alternate turns, placing a piece anywhere on the grid. The first to have at least 5 pieces in a row (horizontally, vertically or diagonally) wins.

We were inspired to create an AI for Connect-5 after both seeing Cris’ Connect-4 parallel AI as well as reading about game state trees for tic-tac-toe. Such zero-sum games are perfect candidates for minimax game state trees with further optimization from α-β pruning. An expansion to Connect-5 as compared to Connect-4 brings the follow challenges we seek to address with parallel approaches:

1) The number of possible moves per turn increases by 20-30 fold depending on the size of the Connect-4 since pieces fall to the bottom in Connect-4 thus making Connect-4 moves 1D compared to the 2D possibilities of Connect-5. This in turn magnifies our search space per ply as well.
2) With a larger search space, there will be more room for optimization, with possibly different approaches at different times in the game (early vs. late), including tree-ordering, α-β pruning and other possible techniques
3) Extending the winning sequence by 1 results in the possibility of more complex heuristics to score each game state, allowing for various strategies including trap-setting such a double, open-ended 3’s

Minimax Game Tree with Heuristics



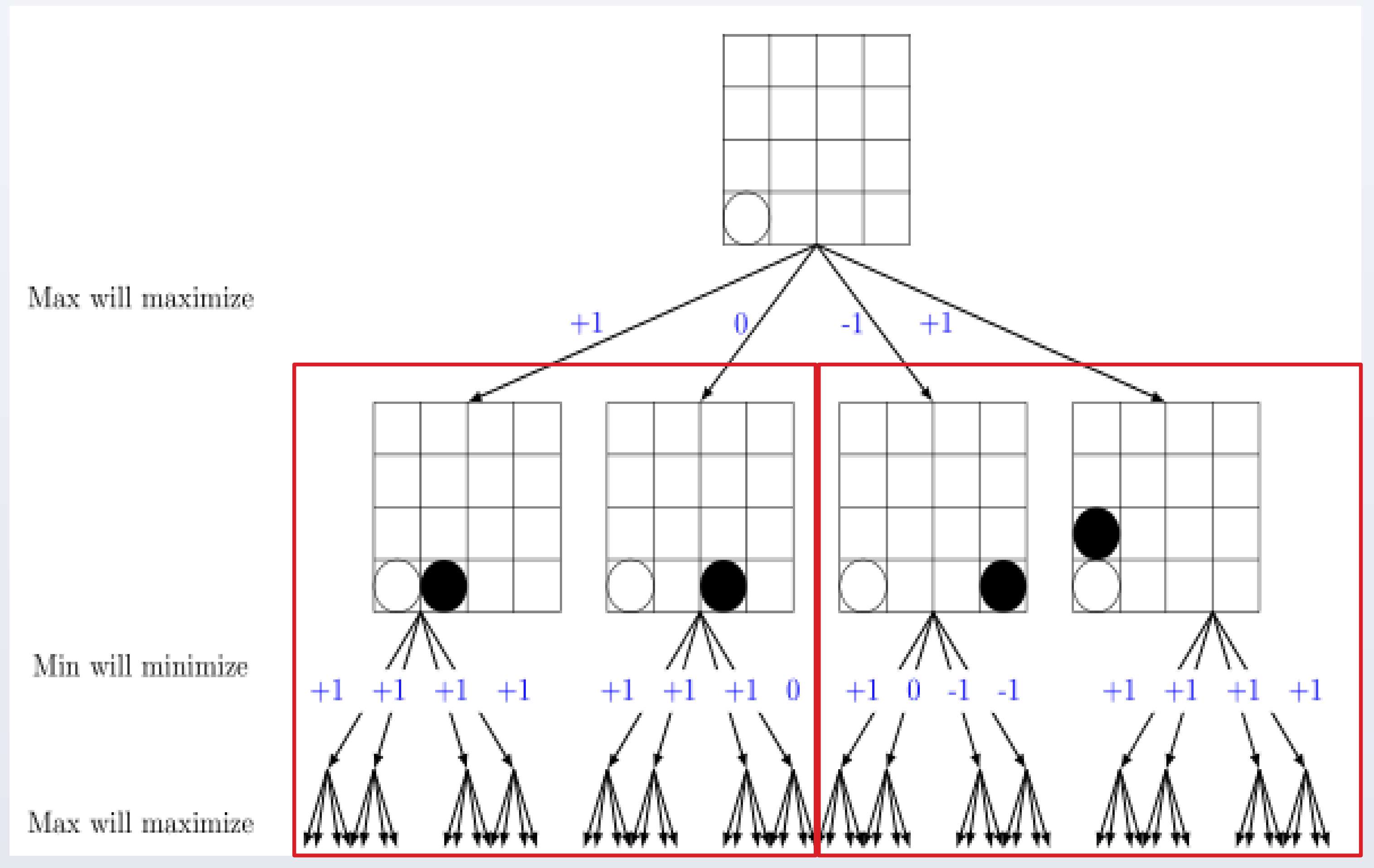
Current Heuristic: “X-in-a-row”

We evaluate (efficiently) the number of 1-in-a-row, 2-in-a-row, 3-in-a-row, 4-in-a-row, and 5-in-a-rows (game over) for each player on the board. We then evaluate the utility by taking the dot product of this vector of numbers with another vector of exponentially increasing weights, to weigh longer combos higher than shorter combos. This heuristic can be adjusted but we feel with proper weighting of the X-in-a-row for scoring, this heuristic will be efficient.

The serial minimax tree essentially maximizes this score, no matter what the heuristic is, assuming that the opponent will play optimally.

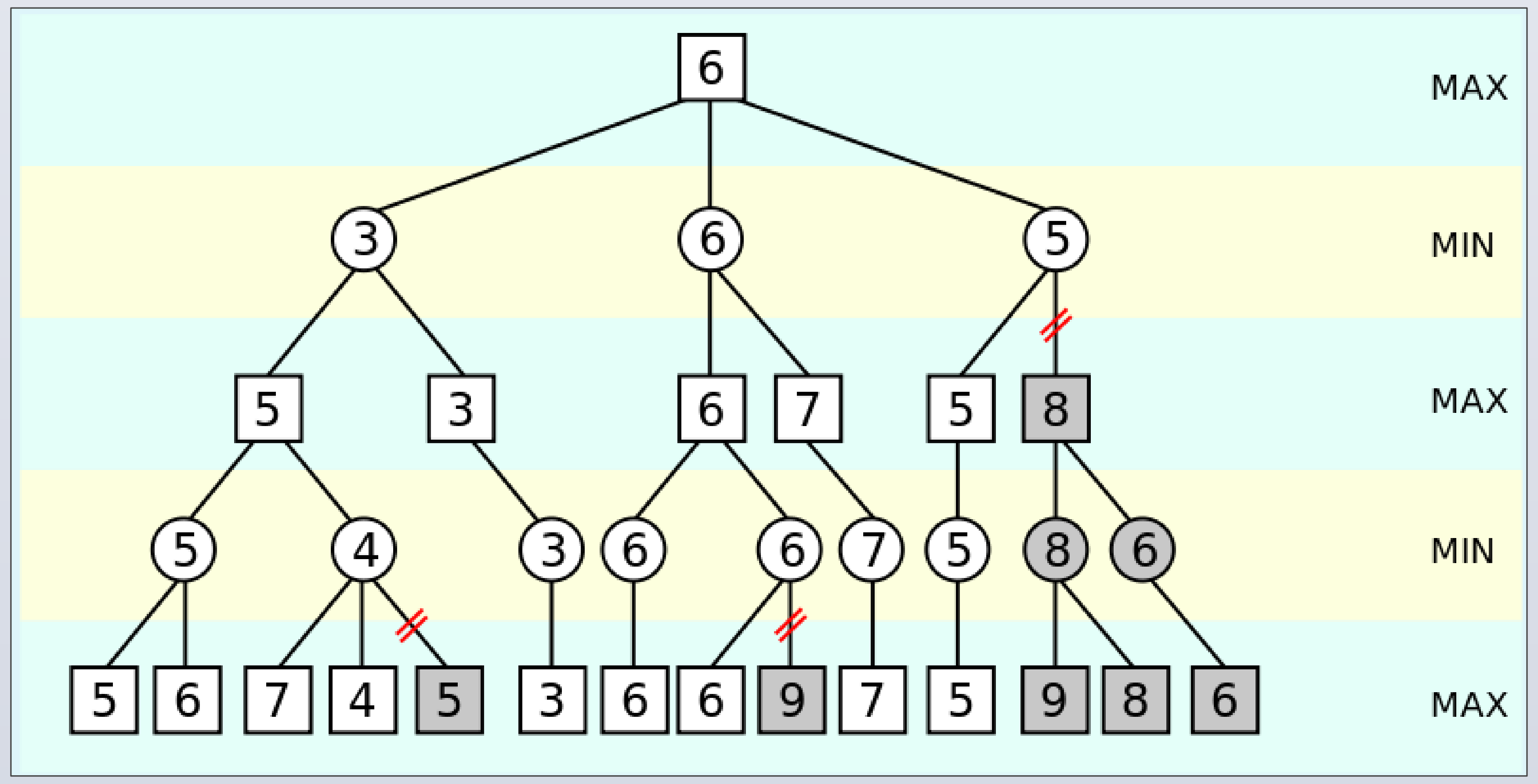
Parallelizing Minimax and Heuristic Computations

A minimax game tree is an embarrassingly parallel problem. For each ply, one more depth of all possible moves must be evaluated. If we do not consider any optimization, which we will not in this section, our full game tree can be evenly divided if we break off the work at the first layer. Thus, each processor will handle a subset of depth (max-1) trees, or in other words, each processor will examine all the future moves based on fixing a subset of the immediate next move. We plan to use MPI communication. Master-slave approaches may be implemented but with a pure minimax tree, the load will always be balanced and therefore scatter/gather methods may be simpler to implement.



In principle, the heuristic computation could also be done by GPU but because of the nature of the various heuristics – only needing to check the immediate squares affected by the last move played, the actual heuristic computation is not as time consuming as simply computing over the space of possible game states.

Optimization: α-β Pruning



α-β pruning is the standard optimization technique for minimax game state trees. The algorithm updates the α-β values which represent the minimum of the maximizing player and the maximum of the minimizing player. All branches that are outside of these assured bounds need not be evaluated.

Tree ordering, or evaluating the best moves first, will better prune the tree. – in other words the branches with the greatest pruning power will be evaluated first. It has been shown that with optimal ordering, the same computation power will allow for twice the number of ply searched .

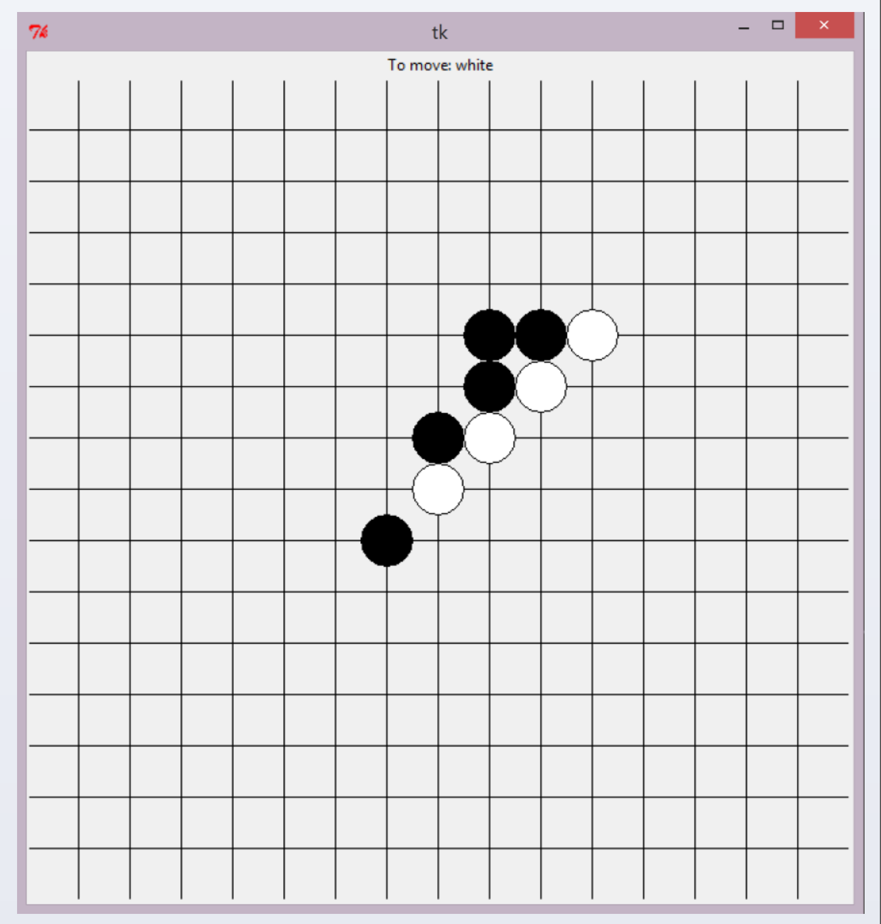
Parallelizing Complexity Reduction and Optimization

Parallelizing search space reduction is an area that we will experiment with. The easiest technique would be to run this optimization portion serially, - meaning each processor will use the optimization algorithm from its own first node and below. This should give a speedup over the non-pruned version. However, this setup will not allow for the information on one branch to prune the other branches. Complete α-β pruning is by nature, a serial algorithm as the more that is searched serially allows for more pruning to follow. Below are a couple of methods we have considered to parallelize α-β pruning if necessary:

1) One processor will be dedicated to sampling evenly from all the processor’s branches a set of states from which it will constantly update and broadcast the α-β values that will be used for pruning – this will not always result in the mot optimal α-β to prune with but will allow the most parallel work to be done
2) Each processor will asynchronously send and receive current best α-β values for every other processor to see and use to prune. This should result in full pruning but may have issues with simultaneous updates when two processors try to update the “new” best α-β values

Current Work

We are currently in the midst of parallelizing the minimax game tree as well as planning how we chose to parallelize the search space reduction portion. The first step will be to compare the parallelize minimax to the serial version. For reference, our serial version, thinking just 2 moves ahead takes 10 seconds per turn pre search-space reduction. We hope to use these numbers to calculate our efficiency for the parallel version – which we expect something close to 1 – as we are perfectly dividing the number of nodes across the processors if we do not prune the tree beforehand.



Future Work

There are many ideas to pursue in optimizing our AI. We recognize that with the limitations on pruning with our parallel case, a well-designed (tree ordering and α-β pruning) may result in faster computations of the game tree during the early stages or when the game board is densely packed. We would be interested in creating a heuristic in which the AI would determine whether parallel computation would be faster or of serial, full pruning would be faster.

We would also like to amortize our game state trees – in which the same game state resulting from a different series of moves would be identified as the same. A better amortization would even account for symmetry, as our game as 2 reflective and 3 rotational symmetries other than the identity. This would significantly reduce the computation and would be an ultimate goal, though we find its implementation non-trivial.

Finally, the nature of the minimax algorithm always assumes perfect play by the opponent and does not capitalize on possible mistakes or traps that a human opponent could make. Thus it plays very passively but a more aggressive algorithm would deviate from the well-studied minimax and is also non-trivial.

References

Rivest, R. Game Tree Searching by Min/Max Approximation.

Stuart, R, Norvig, P. Artificial Intelligence: A Modern Approach (3rd ed).

Acknowledgments

We would like to thank the staff of CS205 headed by Prof. Cris Cecka in giving us the opportunity to pursue this project. We would also like to thank IACS for the chance to present the ongoing work of this project.